Inteligencja obliczeniowa - Reinforcement Learning

Grzegorz Madejski

Na poprzednim wykładzie...

- Omówiliśmy, że większość czynności/procesów da się opisać jako *decyzyjny proces Markova* (MDP) tzn. czynności wykonywane są w krokach, agent co krok bada środowisko (aktualne obserwacje) i na bazie obserwacji podejmuje akcje, za które może dostać nagrody.
- Agent działa wg wyuczanej strategii (*polityka*). Celem algorytmów jest znalezienie najlepszej polityki.
- Pokazaliśmy środowiska wirtualne *Gym* m.in gry LunarLander czy FrozenLake.

Czy da się wyuczyć sieć neuronową do tych zadań?

- Sieć neuronowa potrzebuje inputów i sensownych outputów, by w procesie uczenia nadzorowanego (algorytmem wstecznej propagacji) poprawić swoje wagi.
- Nie mamy do dyspozycji bazy danych z takimi danymi. Inputy (obserwacje środowiska) łatwo wygenerować. Ale sensownych outputów nie ma.
- Sieci neuronowe (uczące się w klasycznym sensie) nie za abrdzo nadają się do tego zadania.
- Trzeba rozpatrzyć algorytmy uwzględniające nagrody za akcje. Mamy tutaj dwie kategorie takich algorytmów. Jedne z *programowania dynamicznego*, a drugie z *programowania przez wzmacnianie* (reinforcement learning).

DP vs RL

Jaka jest różnica pomiędzy DP i RL?

- *Klasyczne algorytmy dynamiczne* (dynamic programming). Zakładają, że znamy model (prawdopdobieństwa przejścia ze stanu do stanu za pomocą wybranej akcji). Wówczas szukanie rozwiązania, polega jedynie na wyszukiwaniu odpowiedniej sekwencji ruchów w tym modelu.
- *Algorytmy uczące przez wzmacnianie* (ang. reinforcement learning, RL). Zakładają, że modelu nie znamy. Model jest tworzony na bieżąco, podczas eksplorowania środowiska. Prawdopodobieństwa są obliczane i poprawiane w iteracjach algorytmu.

Programowanie dynamiczne

Programowanie dynamiczne

Idea DP

Jaka jest ogólna zasada działania algorytmów DP?

- Celem algorytmów DP jest znalezienie optymalnej polityki, czyli sekwencji akcji maksymalizującej sumę nagród po dotarciu do stanu terminalnego.
- Skoro znamy model tranzycji, to możemy wybierać takie akcje, które prowadzą do najlepszych nagród...
- Pytanie: czy wybieramy zawsze najlepszą nagrodę w aktualnym kroku? Czy próbujemy przewidzieć kolejne nagrody?

Nagrody

- Przypomnienie: po wykonaniu każdej akcji a prowadzącej ze stanu s do stanu s' algorytm otrzymuje natychmiastową *nagrodę* (ang. reward) $R_a(s, s')$. Nagroda może być karą (ang. penalty) jeśli ma wartość ujemną.
- Czy powinniśmy być zachłanni i wybierać zawsze największą nagrodę dla bieżącej akcji?

Zwroty i dyskonty

- Należy patrzeć w przyszłość i oszacować, czy *zwrot* (ang. return) takiej sekwencji akcji nam się opłaca. Przez zwrot rozumiemy sumę przyszłych nagród.
- Który zwrot lepszy: 3 czy 1+1+1+1+1?

Zwroty i dyskonty

- W praktyce jednak bywa, że sekwencje akcji są zapętlone i nieskończone i zwrot wyszedłby nam $+\infty$.
- Dlatego wprowadza się *współczynnik dyskontowania* (ang. discount) $\gamma \in [0, 1]$, który czyni takie nieskończone sumy zbieżnymi.
- Jeśli $\gamma = 0.9$, to pies z obrazka oszacuje nagrodę na $1 + 0.9 \cdot 1 + (0.9)^2 \cdot 1 + (0.9)^3 \cdot 1 + (0.9)^4 \cdot 1 = 4.0951$ (strategia cierpliwa, dalekowzroczna)
- Jeśli $\gamma = 0.1$, to pies z obrazka oszacuje nagrodę na $1 + 0.1 \cdot 1 + (0.1)^2 \cdot 1 + (0.1)^3 \cdot 1 + (0.1)^4 \cdot 1 = 1.1111$ (strategia niecierpliwa, krótkowzroczna)
- W naturze ludzi i zwierząt leży myślenie, że odległe nagrody są mniej atrakcyjne i mniej pewne.

Zwroty i dyskonty

Oczywiście inną taktyką na liczenie zwrotu jest ograniczenie się do skończonej liczby akcji w przyszłość. Mówimy wówczas o *skończonym horyzoncie*.

- Zwrot z nieskończonym horyzontem i dyskontem γ:

$$G_t = R_{a_t}(s_t, s_{t+1}) + \gamma R_{a_{t+1}}(s_{t+1}, s_{t+2}) + \cdots = \sum_{i=0}^{\infty} \gamma^i R_{a_{t+i}}(s_{t+i}, s_{t+i+1})$$

- Zwrot ze skończonym horyzontem T i bez dyskontu:

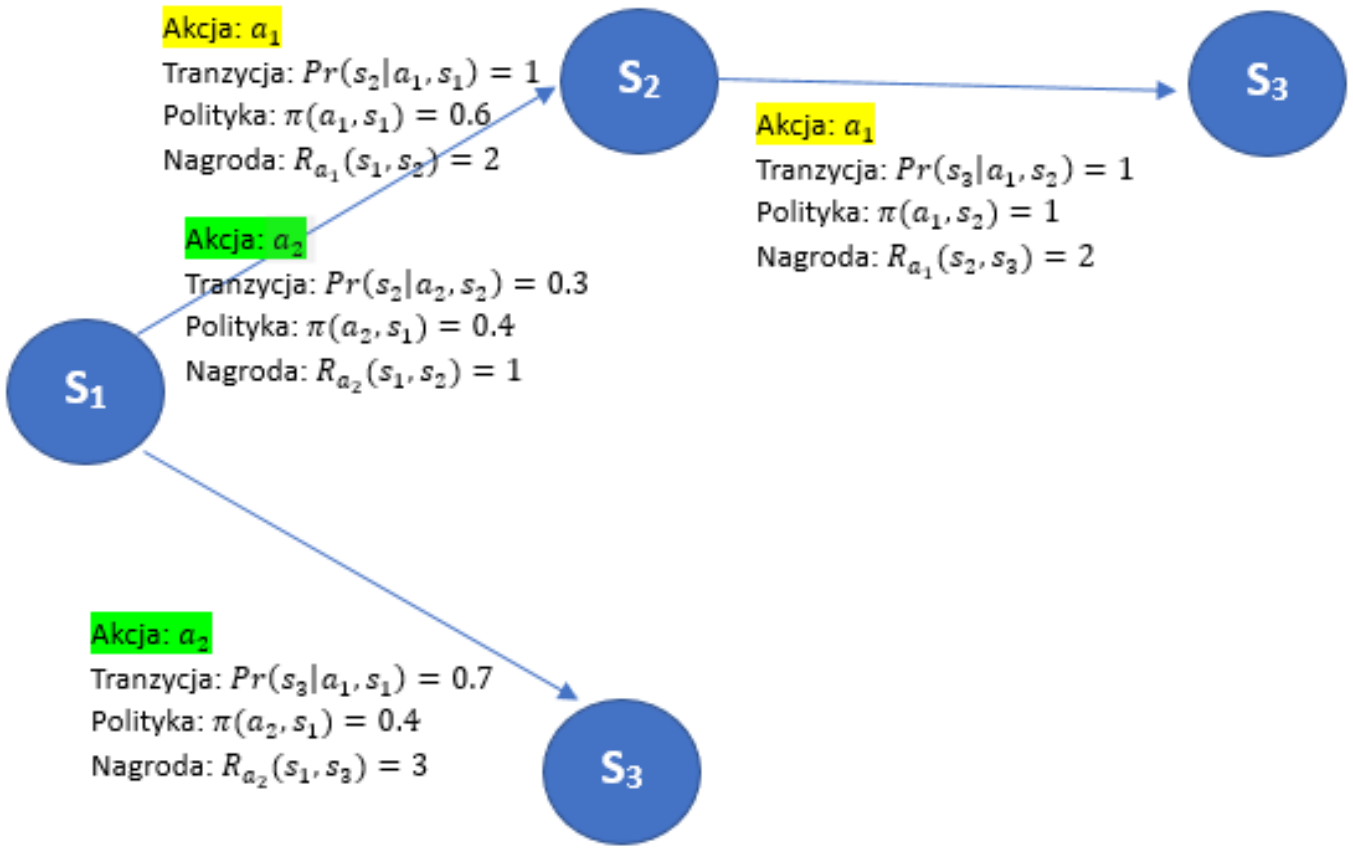
$$G_t = R_{a_t}(s_t, s_{t+1}) + \cdots + R_{a_{t+T}}(s_{t+T}, s_{t+T+1}) = \sum_{i=0}^{T} R_{a_{t+i}}(s_{t+i}, s_{t+i+1})$$

Funkcja wartości stanu

- Zwrot G_t jest liczony dla konkretnego zestawu stanów i akcji.
- Problem jest następujący: nie wiemy, czy dane akcje poprowadzą nas do konkretnych stanów (są przecież poślizgnięcia, turbulencje, porywy wiatru!).
- Polityka może też traktować akcje z różnymi wagami. Premiować jedne, zakazywać innych.
- Będąc w stanie s_1 musimy rozpatrzyć całe drzewo różnych zwrotów zaczynających się od stanu s_1. Dopiero rozpatrzenie całego drzewa daje obraz tego, czy ten stan jest "opłacalny".

Funkcja wartości stanu

Przykład drzewa ze stanem początkowym s_1 i terminalnym s_3. Przeanalizuj je.



Funkcja wartości stanu

- Każda ścieżka w tym drzewie, będąca jednym zwrotem, ma swój zestaw nagród, i też prawodpodobieństwo zajścia (tranzycja, polityka).
- Należy o to uśrednić tj. obliczyć wartość oczekiwaną wszystkich zwrotów zaczynających się w danym stanie.
- Służy do tego specjalna funkcja oceny zwana *funkcją wartości stanu* (ang. *state value function*):

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

Funkcja wartości stanu

Przyjrzyjmy się jak dokładniej rozumieć wartość oczekiwaną z poprzedniego slajdu:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Gdyby brać pod uwagę tylko nagrody w najbliższym kroku to sensowny wzór:

$$V_\pi(s) = \sum_{a \in A} \pi(a, s) \sum_{s' \in S} Pr(s'|a, s) R_a(s, s')$$

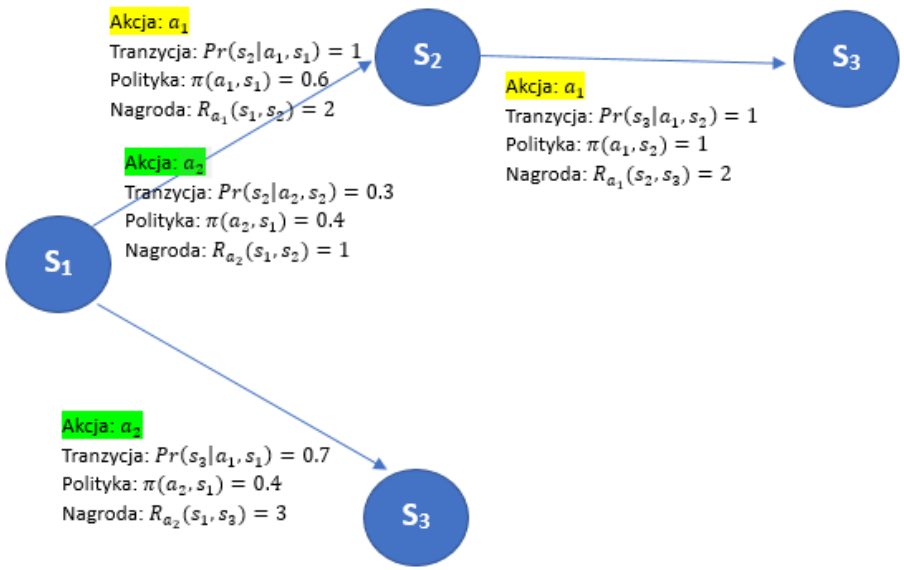
Musimy jednak rekurencyjnie zajrzec w kolejne stany, więc wzór staje się rekurencyjny i jest następujący:

$$V_\pi(s) = \sum_{a \in A} \pi(a, s) \sum_{s' \in S} Pr(s'|a, s) \left(R_a(s, s') + \gamma V_\pi(s')\right)$$

Funkcja wartości stanu

Zadanie

Wykorzystaj wzór $V_\pi(s) = \sum_{a \in A} \pi(a, s) \sum_{s' \in S} Pr(s'|a, s) \left(R_a(s, s') + \gamma V_\pi(s')\right)$ z dyskontem $\gamma = 0.9$ i policz $V_\pi(s_1)$ oraz $V_\pi(s_2)$ dla rysunku. Przyjmujemy, że stan terminalny s_3 ma $V_\pi(s_3) = 0$



Funkcja wartości stanu

Przyjrzyjmy się jak dokładniej rozumieć wartość oczekiwaną z poprzedniego slajdu:

$$V(s) = \mathbb{E}[G_t | S_t = s]$$

Gdyby brać pod uwagę tylko nagrody w najbliższym kroku to sensowny wzór:

$$V_\pi(s) = \sum_{a \in A} \pi(a, s) \sum_{s' \in S} Pr(s'|a, s) R_a(s, s')$$

Musimy jednak rekurencyjnie zajrzec w kolejne stany, więc wzór jest następujący:

$$V_\pi(s) = \sum_{a \in A} \pi(a, s) \sum_{s' \in S} Pr(s'|a, s) \left(R_a(s, s') + \gamma V_\pi(s')\right)$$

Wzór ten nazywamy równaniem Bellmana (dla oczekiwanej nagrody w MDP).

Funkcja wartości akcji

- Drugą (pokrewną) funkcją oceniania jest *funkcja wartości akcji* (ang. *action-value function*) daną wzorem jest wzorem:

$$Q_\pi(s,a) = \mathbb{E}_\pi(G_t|S_t = s, A_t = a)$$

- Wartość oczekiwaną można rozumień nastepująco

$$Q_\pi(s,a) = \sum_{s' \in S} Pr(s'|a,s)\left(R_a(s,s') + \gamma V_\pi(s')\right)$$

- Biorąc wzór z poprzedniego slajdu można otrzymać fajną zależność:

$$V_\pi(s) = \sum_{a \in A} \pi(a,s) Q_\pi(s,a)$$

Szukanie optymalnej polityki

- Szukamy optymalnej polityki $\pi^* : A \times S \longrightarrow [0, 1]$, $\pi^*(a, s) = Pr(a|s)$. Optymalna = daje najlepszą sumaryczną nagrodę.
- Optymalna polityka to taka, dla której V_π jest maksymalne tzn. $V_{pi^*}(s) = \max_\pi(V_{pi^*}(s))$.
- By znaleźć najlepszą politykę należy rozwiązać *równanie optymalizacyjne Bellmana*:

$$V_{\pi^*}(s) = \max_a\{\sum_{s' \in S} Pr(s'|a, s)\left(R_a(s, s') + \gamma V_{\pi^*}(s')\right)\}$$

Algorytmy programowania dynamicznego

Jeśli chodzi o algorytmy dynamiczne szukające najlepszej polityki, to są dwa popularne:

- Iterowanie po stanach (ang. value iteration algorithm)
- Iterowanie po politykach (ang. policy iteration algorithm)

Algorytm: Iterowanie po stanach

Algorytm iterujący po wartościach stanów (ang. *state value iteration algorithm*) oblicza każdemu wierzchołkomi optymalną ocenę (wartość $V(s)$) biorąc pod uwagę nagrody jakie może zdobyć. Optymalna deterministyczna polityka π będzie wybierać takie akcje, żeby trafiać do stanów o jak największym $V(s)$. Uwaga! Algorytm wymaga pełnej wiedzy o środowisku (wszystkie stany).

Algorytm: Iterowanie po stanach

1. Dla każdego $s \in S$ ustaw losowe $V(s)$.
2. Dla każdego stanu $s \in S$:
   - $temp(s) = V(s)$
   - Dla wszystkich $a \in A$ policz:
     $Q(s,a) = \sum_{s' \in S} Pr(s'|a,s)\left(R_a(s,s') + \gamma V(s')\right)$
   - $V(s) = \max_a\{Q(s,a)\}$
   - $\pi(s) = \text{argmax}_a\{Q(s,a)\}$
3. Czy dla każdego stanu s, $|V(s) - temp(s)|$ jest dostatecznie małe?
   TAK $\rightarrow$ krok 4.
   NIE $\rightarrow$ idź do kroku 2.
4. Zwróć optymalną politykę deterministyczną π.

Algorytmy programowania dynamicznego

- Algorytm iterujący po stanach zawsze się kończy zbiegając do optymalnego zestawu wartości V, który wyznacza optymalną politykę deterministyczną π^*.
- Podobnym algorytmem jest *algorytm iterujący po politykach*, który zbiega do optymalnej polityki. Go pomijamy na wykładzie.
- Więcej można poczytać np. tutaj: `https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration`

Uczenie przez wzmacnianie

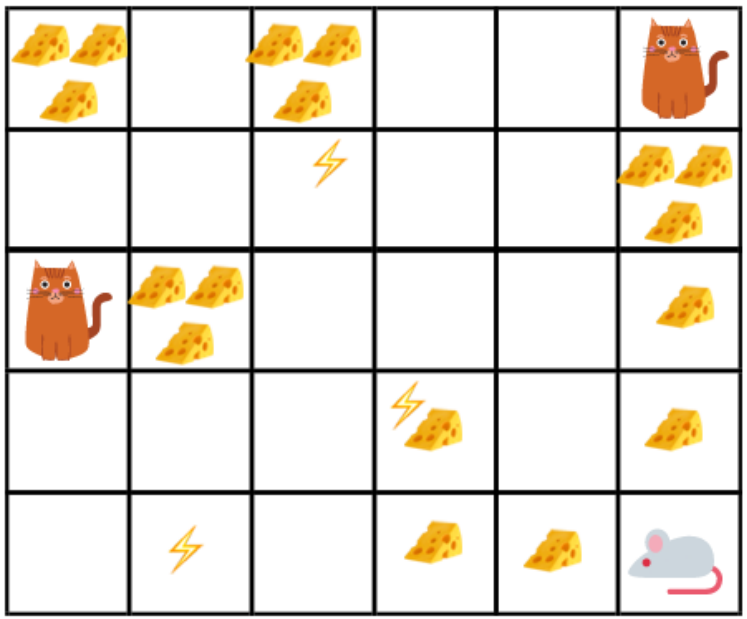
Uczenie przez wzmacnianie (Reinforcement Learning)

Uczenie przez wzmacnianie

- *Uczenie przez wzmacnianie* (ang. *Reinforcement Learning*, RL) to szereg technik i algorytmów szukających optymalnych strategii na bazie doświadczenia.
- W przeciwieństwie do alg. programowania dynamicznego, nie trzeba mieć pełnej wiedzy o modelu/środowisku! Wiedzę o środowisku zdobywamy poprzez doświadczenie.
- Mamy więc dwa zadania do wykonania: zdobywać doświadczenie przez eksplorację środowiska (exploration) i wykorzystać doświadczenie do znalezienia optymalnego rozwiązania (exploitation).
- Pomiędzy realizacją obu celów trzeba znaleźć zdrowy balans/kompromis.

Uczenie przez wzmacnianie

Czy eksploracja jest ważna? Czy lepiej zawsze iść tam gdzie jest ser?



Źródło: `https://www.freecodecamp.org/news/a-brief-introduction-to-reinforcement-learning-7799af5840db`

Typy RL

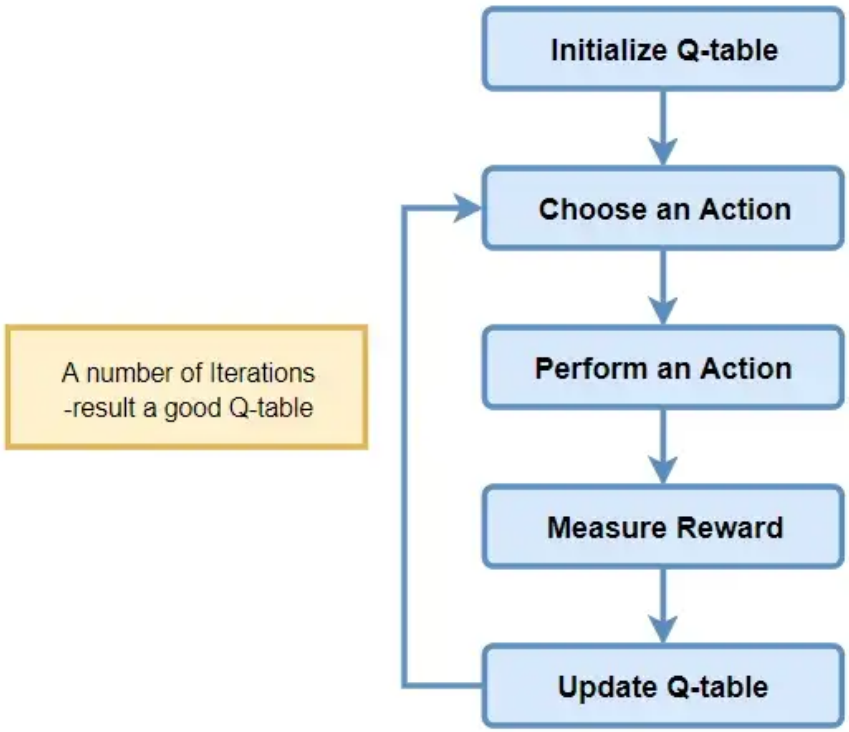
- Jest wiele algorytmów RL. Algorytmy mają różne cechy.
- Rozróżniamy algorytmy typu *off-policy* i *on-policy*. Pierwsze tworzą politykę poprzez wykonywanie różnych ruchów, drugie wykonują swoją politykę, oceniają ją i poprawiają.
- Są algorytmy *model-free* i *model-based*. Pierwsze obliczają politykę na podstawie doświadczeń. Drugie tworzą model świata, tzn uczą się jak wygląda model tranzycji i nagrody za akcje, dzieki czemu mogą przewidywać przyszłe ruchy i planować.
- Różne algorytmy nadają się też do różnych zadań. Akcje mogą być *dyskretne* (kilka do wyboru) lub *ciągłe* (nieskończenie wiele). Stan środowiska również może być dyskretny lub ciągły.

Q-Learning

- *Q-Learning* to algorytm uczenia przez wzmacnianie, wykorzystujący tabelę jakości (zwaną *Q-table*, quality table), gdzie przechowujemy opłacalność akcji w danych stanach.
- Tabela ma wymiary $|S| \times |A|$. W komórce $Q[s, a]$ stoi aktualnie obliczona opłacalność akcji a w stanie s.
- $Q[s, a]$ jest oczekiwaną wartością (skumulowany dyskontowany zwrot) wykonania akcji a w stanie s.
- Algorytm po wielu iteracjach zbiega (converge) do optymalnej polityki. Polityka: dla s wybieramy akcję a, która daje największe $Q[s, a]$.
- Q-learning jest algorytmem typu off-policy, model-free dla dyskretnych akcji i stanów.

Q-Learning

Schemat działania algorytmu źródło: `https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c`):



Q-Learning

Jak wygląda Q-table? Przykład: FrozenLake4x4.

Q-Table Frozen Lake				
	←	↑	→	↓
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Q-Table Frozen Lake (Wersja mniejsza)				
	←	↑	→	↓
PoleStart				
PoleLód				
PoleDziura				
PoleCel				

Q-Learning

Wzór na poprawę Q-table, gdy w stanie s wybrano akcję a i przeszliśmy do stanu s':

$$Q[s,a] = Q[s,a] + \alpha \left(R[a,s,s'] + \gamma \cdot max_{a'} Q[s',a'] - Q[s,a] \right)$$

- Bierze starą wartość $Q[s,a]$ i dodaje do niej pewien nowy składnik.
- Składnik ten mnożony jest przez *współczynnik uczenia się* α (learning rate).
- Składnik składa się on z *nagrody* $R[a,s,s']$, *maksymalnej skumulowanej nagrody w przyszłej akcji* $max_{a'} Q[s',a']$ *zdyskontowanej przez* γ. Pomniejszony o aktualne $Q[s,a]$.

Q-Learning

Algorytm Q-learning (hiperparametry: α, γ, ϵ):

1. Zainicjuj tablicę $Q[s, a]$ (zwykle z zerami).
2. Powtórz dla każdego epizodu:
   - Reset środowiska. Przejdź do stanu startowego $s = s_0$.
   - Powtórz dla każdego kroku w epizodzie (aż do terminacji):
     - Wybierz akcję a z aktualnego stanu s (uwzględnij w tym losowość i/lub Q-tabelę[1])
     - Uruchom akcję a, i sprawdź wynik r i s'
     - $Q[s, a] = Q[s, a] + \alpha(r + \lambda \cdot max_{a'} Q[s', a'] - Q[s, a])$
     - $s = s'$
3. Oblicz i zwróć optymalną politykę: $\pi(s) = argmax_a Q[s, a]$.

[1]Akcję możemy wybrać metodą *epsilon greedy policy* tzn. jeśli random wypadnie poniżej jakiejś wartości ϵ to wybierz losową akcję, a w przeciwnym wypadku wybierz $argmax_a Q[s, a]$

Q-Learning

Różne linki, źródła dla Q-Learning:

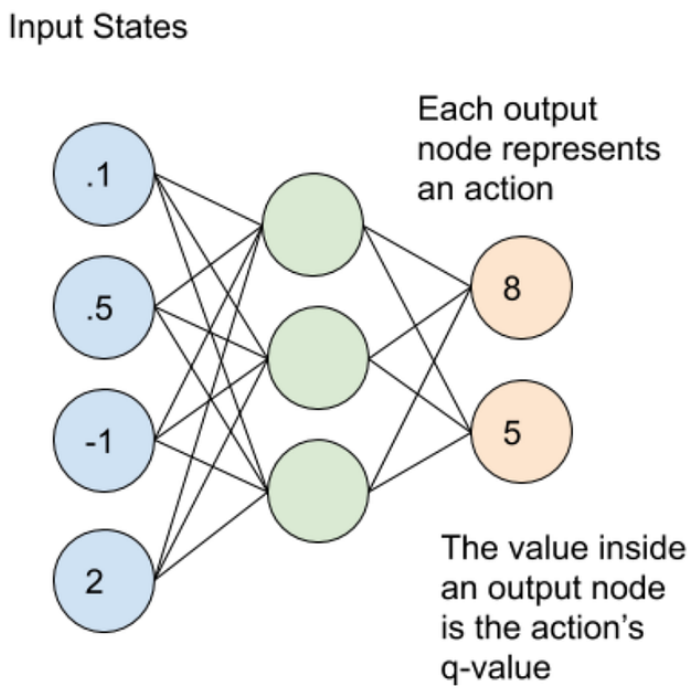
- Q-Learning: `https://en.wikipedia.org/wiki/Q-learning`
- Q-learning i FrozenLake: `https://towardsdatascience.com/q-learning-for-beginners-2837b777741`
- Q-learning examples: `https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/`
- Q-learning i FrozenLake: `https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial`
- Q-learning and blackjack: `https://towardsdatascience.com/reinforcement-learning-solving-blackjack-5e31a7fb371f`

Deep Q-Network

- Podstawowy algorytm Q-learning wykorzystuje tabelkę dla stanów i akcji, przez co ograniczony jest do przestrzeni dyskretnej.
- Dla przestrzeni ciągłej stanów i dyskretnej akcji, można wykorzystać sieć neuronową.
- Sieć neuronowa dostaje na wejściu stan środowiska (zestaw liczb zmiennoprzecinkowych), ma tyle neuronów wyjściowych ile akcji, a na każdym neuronie wyjściowym oblicza q-wartość dla tej akcji (i stanu wejściowego).
- Taką sieć nazywamy *Deep Q-Network* i jest to algorytm off-policy, model-free.

Deep Q-Network

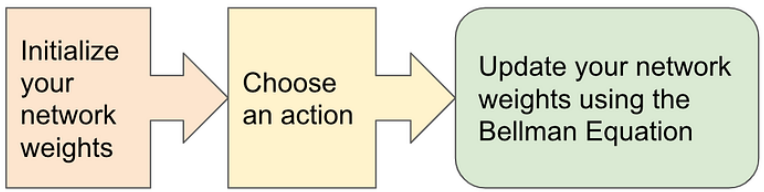
Przykład sieci:



Źródło: `https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc`

Deep Q-Network

Jak działa algorytm?



Źródło: `https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc`

Deep Q-Network

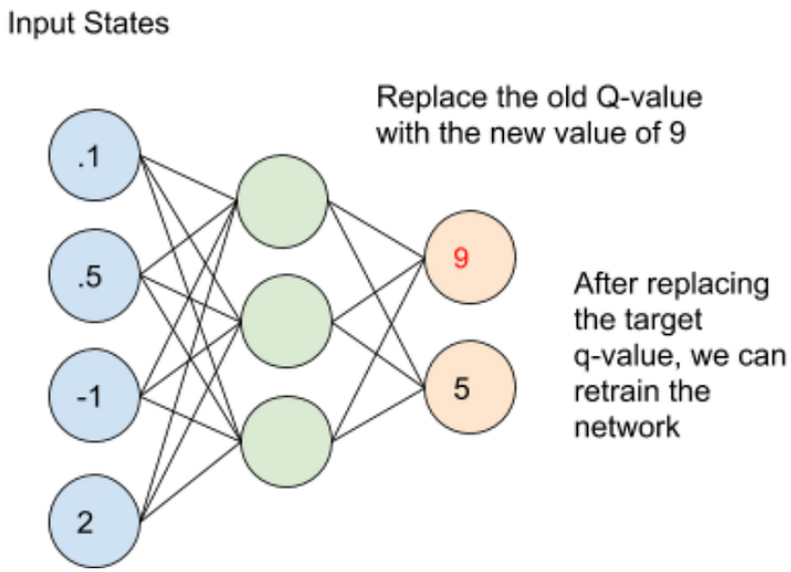
Analogicznie jak w Q-learning, należy wykorzystać równanie Bellmana aby poprawić q-wartości.

$$(R_t + \lambda * max_a Q(S_{t+1}, a))$$

Źródło: `https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc`

Deep Q-Network

Następnie trenujemy sieć (poprawiamy wagi) by dopasowały się do nowego wyniku.



Źródło: `https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc`

Linki

Inne przydatne linki do wykładu:

- Wykład z MDP: `https://www.cs.put.poznan.pl/wjaskowski/pub/teaching/wmio/lectures2014/Problemy_Decyzyjne_Markova.pdf`
- Wzory i definicje `https://medium.com/intro-to-artificial-intelligence/key-concepts-in-reinforcement-learning-2af715dfbfa`
- Fajne wykłady z RL: `https://www.davidsilver.uk/teaching/`
- Algorytmy dynamiczne: `https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration`
- Funkcje oceny: `https://medium.com/intro-to-artificial-intelligence/relationship-between-state-v-and-action-q-value-function-in-reinf`
- Deep Q Network: `https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc`